RCE User Guide

Build 6.3.0.201507291548

Table of Contents

1. Preface 1
    1. Abstract 1
    2. Intended Audience 1
    3. License Information 1
    4. Compatible Operating Systems 1
        4.1. Support of 32 Bit Operating Systems 2
        4.2. Known Issues 2
2. Setup 3
    1. Installation on Linux 3
        1.1. Prerequisites 3
        1.2. Installation 3
            1.2.1. Debian/Ubuntu/Mint - Installing from the Package Repository (recommended) 4
            1.2.2. Debian/Ubuntu/Mint - Installation of the .deb Package (alternative) 4
            1.2.3. CentOS/Red Hat - Installation of the .rpm Package (recommended) 5
            1.2.4. All Distributions - Running from the .zip File (alternative) 5
        1.3. Digital Signatures and Download Verification 5
        1.4. Starting RCE as a GUI Client 6
        1.5. Starting a Non-GUI ("Headless") Instance 6
        1.6. Installation as a Daemon on a Linux Server 7
            1.6.1. Installation and Daemon Management 7
            1.6.2. Daemon Configuration 7
    2. Configuration 7
        2.1. Configuration Locations and Files 8
        2.2. Configuration Parameters 8
3. Usage 12
    1. Graphical User Interface 12
    2. Running and Creating Workflows 14
    3. Workflow Components 16
        3.1. Execution Scheduling of Workflow Components 16
    4. Commands 17
        4.1. Command Line Parameters 17
        4.2. Command Shell 18
            4.2.1. Configuration Placeholder Value Files 20
    5. Integration of External Tools 22
        5.1. Basic Concepts 22
        5.2. Copying of Integrated Tools 22
            5.2.1. Publishing a Component on a Server Instance in Headless Mode 23
        5.3. Integration of CPACS Tools 23
            5.3.1. Additional concepts of CPACS Tool Integration 23
            5.3.2. Integrate a CPACS Tool into a Client Instance 25
            5.3.3. Integrate a CPACS Tool into a Server Instance in Headless Mode 27
    6. Remote Tool and Workflow Access 27
        6.1. Basic Concepts 27
        6.2. Prerequisites 27
        6.3. Tool vs. Workflow Execution 28
        6.4. Setting up the Single Tool Execution Example 28
        6.5. Setting up the Workflow Execution Example/Template 30
        6.6. Building Your Own Remote Access Workflow 32

List of Figures

3.1. Workbench with different views and the workflow editor opened 12
3.2. Connection Editor 13
3.3. Network View 13
3.4. Workflow Data Browser 14
3.5. Workflow Console 14
3.6. Run process of an user-integrated CPACS Tool 25

List of Tables

2.1. Linux installation options 3
2.2. general 8
2.3. network 9
2.4. publishing 10
2.5. componentSettings 10
2.6. sshServer 10
3.1. Command line arguments for RCE 17
3.2. Shell Commands 19
3.3. Components and their configuration placeholders 21

Chapter 1. Preface

This chapter gives an introduction to RCE.

1. Abstract

RCE (Remote Component Environment) is an open source software to help engineers, scientists and others to create, manage and execute complex calculation and simulation workflows. A workflow in RCE consists of components, which are connected between each other. A component can be a simulation tool, a tool for data access, or a user-defined script. Connections define which data flows from one component to another. There are predefined components with common functionalities, like an optimizer or a cluster component. Moreover, users can integrate their own tools. RCE instances can be connected between each other. Components can be executed locally or on remote instances of RCE (if the component is installed accordingly). Having this at hand, use-cases for complex distributed applications can be solved with RCE.

2. Intended Audience

The intended audience of this document are engineers, scientists, and everybody else interested in developing automated workflows with RCE.

3. License Information

RCE is published under the Eclipse Public Licence (EPL) 1.0. It is based on Eclipse RCP 3.7.2 (Indigo SR2), which is also published under the Eclipse Public Licence (EPL) 1.0. RCE also makes use of various libraries which may not be covered by the EPL; for detailed information, see the file "THIRD_PARTY" in the root folder of an RCE installation. (To review this file without installing RCE, open the RCE release .zip file.)

For downloads and further information, please visit http://www.rcenvironment.de.

4. Compatible Operating Systems

RCE releases are provided for Windows and Linux. It is regularly tested on

- Red Hat Enterprise Linux 6 Workstation (64 bit)
- Debian 7 stable (64 bit)
- SUSE Linux Enterprise Desktop ("SLED") 11 SP2 (64 bit)
- Windows 7 (64 bit)

It is also confirmed to run (but not as thoroughly tested) on

- Mint 17 (64 bit)

- CentOS 7 (64 bit)
- SUSE Linux Enterprise Server ("SLES") 11 SP2 (64 bit)
- Ubuntu 14 LTS (64 bit)
- Windows Server 2008 r2 (64 bit)

4.1. Support of 32 Bit Operating Systems

Starting with release 6.0.0, only the 64 bit versions of RCE are being fully tested. The 32 bit packages are provided for backwards compatiblity and may be removed in a future release. If you rely on up-to-date 32 bit versions of RCE, or encounter a problem with these packages, please contact the RCE developers so we can take this into account.

4.2. Known Issues

On Red Hat Enterprise Linux 7 with KDE 4, RCE (like any other Eclipse-based application) can cause a segmentation fault at startup. If you encounter such an issue, you can try choosing a different GTK2 theme:

1. Open the **System Settings** application (systemsettings).
2. Go to **Application Appearence**
3. Open **GTK** page
4. Switch the GTK2 theme to "Raleigh" or "Adwaita" and click on **Apply**

Chapter 2. Setup

This section describes the installation and configuration of RCE.

1. Installation on Linux

1.1. Prerequisites

To run RCE on a system, the only prerequisite is an installed Java Runtime Environment (JRE), version 7 or above. On Linux, use your system's package manager to install the OpenJDK JRE.

1.2. Installation

Starting with RCE 6.0.0, there are two installation options on Linux: installing RCE from .deb/.rpm packages, or extracting it from a zip file (which was traditionally used by RCE 5.x and before). Whenever possible, using the packages is recommended, as RCE is properly installed into the proper system locations, and can be cleanly managed using your distribution's package manager. If you are on a Debian-based distribution, a package repository is provided as well, which makes it even easier to install and upgrade RCE. The following table compares these options:

Table 2.1. Linux installation options

Installation type	**Multi-user operation supported**	**Daemon operation (system service) supported**	**Installing more than one version at the same time**	**File system location**	**Updating to a new version**	**Verifying digital signatures**	**Registers start menu entry and icon**
Using the package repository *(Debian/ Ubuntu/ Mint only)*	**yes**	**yes**	no	`/usr/share/rce`	Using the distribution's update manager (automatic or manual)	**automatic**	**yes**
Manual installation of the.deb/.rpm package	**yes**	**yes**	no	`/usr/share/rce`	Manually download and install a newer package	manual (gpg/ shasum)	**yes**
Unpacking the .zip file	no	no	**yes**	(anywhere)	Use "Help > Check for Updates" in RCE **-or-** delete the old installation directory and manually download and unpack a newer zip file	manual (gpg/ shasum)	no

1.2.1. Debian/Ubuntu/Mint - Installing from the Package Repository (recommended)

To register the RCE .deb package repository in your system, you have to add the repository location, and add the RCE signing key. Then, you can install RCE like any other software package. These steps can either be done with a graphical tool, or from the command line; both have the same effect.

- If you prefer graphical tools, start "Synaptic"; if it is not available, install if from your Software Center - or similar - first.
  - Open "Settings > Repositories".
  - Click on "Authentication keys", and then on "Download a key...". Copy the text `258BC129EDA2389D3ECD2DE6BA880CB39DC1CE34` into the dialog that appears and click "Ok". The list of keys should now contain an entry "RCE 6.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>".
  - Click on a section named "Additional Repositories", "Other Sources" or similar. Select the "Add..." or "Add new..." option. Copy the text "`deb http://software.dlr.de/updates/rce/6.x/products/standard/releases/latest/deb/ /`" (without the quotes) into the dialog that appears and click "Ok".
  - Close the settings dialog and click "Refresh" to update the list of available software. After this, you should find an entry "rce" in this list (for example, using the "quick search" box), which you can install like any other software.
- If you prefer the command line, enter the following commands as "root" (or with the "sudo" prefix):
  - `apt-key adv --keyserver hkp://keys.gnupg.net:80 --recv-key 258BC129EDA2389D3ECD2DE6BA880CB39DC1CE34`

Note

If you see a message containing the text "`RCE 6.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>`", the key import was successful. Sometimes, this step fails with the message "`key [...] not found on keyserver; gpg: no valid OpenPGP data found.`" If this happens, just repeat the command a few times until it works.

  - `apt-add-repository "deb http://software.dlr.de/updates/rce/6.x/products/standard/releases/latest/deb/ /"`
  - `apt-get update`
  - `apt-get install rce`

Once you have installed RCE using either of these approaches, any RCE 6.x upgrade will automatically show up in you operating system's update mechanism. Depending on the upgrade settings of your system, they may be installed automatically, or be presented to you for selection.

Note

Although technically possible, RCE 6.x will not auto-upgrade to 7.x (or higher) to maintain compatibility within an RCE 6.x network. You will need to install the 7.x repository location to upgrade.

1.2.2. Debian/Ubuntu/Mint - Installation of the .deb Package (alternative)

To install the .deb file manually, download the latest version from `http://software.dlr.de/updates/rce/6.x/products/standard/releases/latest/deb/`. You can either

install it with the graphical package management tools of your distribution (double-clicking the .deb should start them), or by running `sudo dpkg -i <filename>` from a terminal. To upgrade an existing installation, simply install the newer package. The package manager will detect the upgrade and handle it properly.

1.2.3. CentOS/Red Hat - Installation of the .rpm Package (recommended)

To install the .rpm file, download the latest version from `http://software.dlr.de/updates/rce/6.x/products/standard/releases/latest/rpm/`. You can either install it with the graphical package management tools of your distribution (double-clicking the .deb should start them), or by running the appropriate installation command as the "root" user. On CentOS or Red Hat, the command is `yum install <filename>`; on SUSE, use `zypper install <filename>`. To upgrade an existing installation, simply install the newer package. The package manager will detect the upgrade and handle it properly

1.2.4. All Distributions - Running from the .zip File (alternative)

If none of the previous installation options fits your needs, you can also extract RCE from the .zip file.

- If you prefer graphical tools, double-click the .zip file to open it with your distribution's archive manager. Extract it to a location of your choice, and open that location in your file-system explorer. Typically, double-clicking the "rce" executable will work out of the box to start RCE. If it doesn't, right-click the executable, open its "properties" section (or similar), and look for an option to mark it as executable. Confirm the dialog, then double-click it again.
- If you prefer using the command line, use the `unzip` command to extract the zip file to a location of your choice and go to that location. Usually, entering `./rce` will be sufficient to start RCE. In same cases, you may need to make it executable first with the `chmod +x rce` command.

1.3. Digital Signatures and Download Verification

Starting with RCE 6.x, we provide digital signatures for our releases. These are intended to verify the integrity of the downloaded files, as *any* unsigned software can be tampered with while downloading it over an unsecured (HTTP) connection. This is especially important when installing RCE from a user account with root privileges, for example as a system daemon

For each type of release files (.zip, .deb, and .rpm files), a SHA256SUM.asc file is provided next to the actual files. This file contains checksums for each release file, and these checksums are signed with our release key. It is named "RCE 6.x Automatic Signing Key", has the id 9DC1CE34, and is valid until Dec 08, 2015. Its full fingerprint is `258B C129 EDA2 389D 3ECD 2DE6 BA88 0CB3 9DC1 CE34`.

On Windows, you need to install gpg4win () to verify the digital signature. To provide the "shasum" command for checking the hash sums, various freeware tools are available. (More detailed information about this will be added in a future release.)On Linux, the required command-line tools to verify these signatures (gpg and shasum/sha1sum) are usually already installed as part of your distribution. If not, use your system's package manager to install them. To perform the actual verification:

- Execute `gpg --keyserver hkp://keys.gnupg.net:80 --recv-key 258BC129EDA2389D3ECD2DE6BA880CB39DC1CE34` to import the signing key. This only needs to be done once per key, e.g. once for all RCE 6.x releases.

Note

If you see a message containing the text `"RCE 6.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>"`, the key import was successful. Sometimes, this step fails with the message

`"key [...] not found on keyserver; gpg: no valid OpenPGP data found."` If this happens, just repeat the command a few times until it works.

- Download the `SHA256SUMS.asc` file from the same location as the installation package. Run `gpg -v --yes SHA256SUMS.asc` in the location where you saved it to; this verifies the digital signature. Inspect the output to see if it is correct; you should find the text "Good signature from "RCE 6.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>" (or a similar translation).

Note

When following these steps, it is normal to receive a warning about the fact that the owner of this key cannot be verified. If you have received this user guide from a trustworthy source (e.g. an official RCE project site secured with HTTPS), you can assume that the key is correct, as the command used to import the key has already verified the key's integrity. Alternatively, you can fetch the key's fingerprint from a trustworthy source (e.g. a secure intranet page, or the official @rcenvironment Twitter feed accessed via HTTPS) and compare it with the one shown in the command's output. If they match, you can trust that you are using the authentic key.

- Run `shasum -c SHA256SUMS` in the same folder. This verifies that the actual download matches what has been digitally signed for the release. You should see the installation package's name, followed by "OK".

Note

On some distributions, you need to use `sha1sum` instead of `shasum`.

1.4. Starting RCE as a GUI Client

Once your RCE instance has started, you can open the configuration file with the menu option "Help > Open Configuration File". Edit the file, save it, and then restart RCE using the "File > Restart" menu option to apply the changes. There are configuration templates and other information available via the "Help > Open Configuration Information" option. The available configuration settings are described in the configuration chapter.

Note

On Ubuntu, the Ubuntu overlay scrollbars can sometimes lead to problems with the RCE GUI. To avoid these problems, you can start RCE from a terminal with `env LIBOVERLAY_SCROLLBAR=0 ./rce` to disable the overlay scrollbars for RCE. Alternatively, if you want to enable the overlay scrollbars permanently for all programs, execute `echo "export LIBOVERLAY_SCROLLBAR=0" > /etc/X11/Xsession.d/80overlayscrollbars` as a superuser and then restart your computer.

1.5. Starting a Non-GUI ("Headless") Instance

RCE can also be run from the command line without a graphical user interface (which is called "headless" mode), which uses less system resources and is therefore recommended when the GUI is not needed. For ad-hoc or temporary RCE network setups, running RCE this way is perfectly fine. For more permanent installations, however, we recommend installing RCE as a system daemon instead. Thas the advantage that RCE automatically shuts down when the server is shut down, and automatically restarts when the server does.

To run a headless RCE instance, open a console and run the command `./rce --headless -console -nosplash`. While RCE is running, you can enter various console commands described in Section 4, “Commands”; note that you need to prefix all RCE commands with "rce " here. To perform a clean shutdown, for example, type `rce stop` and press enter.

1.6. Installation as a Daemon on a Linux Server

1.6.1. Installation and Daemon Management

Executing the following steps will install RCE as daemon. An RCE daemon will start automatically on system boot and stop before system shutdown.

The recommended (and supported) way to install a Linux daemon is to install RCE from a .deb or .rpm package, or (equivalently) installing it from the APT repository. While installing a daemon from the .zip file distribution can be made to work, it creates unnecessary complications regarding installation paths and file permissions. As registering a daemon requires root privileges anyway, there should be no reason to use the .zip file; if you have a compelling use case for this, please contact the RCE team.

Once RCE is properly installed, registering it as a daemon is very simple. A command named `rce-daemon` is provided to control the whole daemon life cycle. The only manual step you need to perform before is creating a user account that the daemon will run under. For now, this account has to be named `rce-daemon`; use you distribution's console commands or GUI tools to add it. (Please note that this account needs to have a home directory to hold the daemon's profile directory.) Future RCE versions will make this account name customizable.

Once this user account is created, you can use the following commands to manage the daemon installation:

- `rce-daemon install` - installs and starts the daemon instance
- `rce-daemon uninstall` - stops and uninstalls the daemon instance; the profile directory remains unchanged
- `rce-daemon start` / `rce-daemon stop` / `rce-daemon restart` - standard daemon controls
- `rce-daemon locate` - prints the location of the daemon's RCE instance's profile directory and relevant files

1.6.2. Daemon Configuration

After installation, the daemon instance will be automatically started. This will create a default configuration file if it does not exist yet.

To configure the daemon instance, use `rce-daemon locate` to find its configuration file, edit and save it, and then use `rce-daemon restart` to apply the new configuration.

Note

The need to restart the daemon is temporary; future versions of RCE will apply configuration changes as soon as the configuration file is changed.

2. Configuration

This section describes the configuration of RCE. Configuration is done within one single configuration file. It is located in the profile directory. From the graphical user interface, you can easily access it from the tool bar or *Help* menu. Note: To apply changes you need to restart RCE. The format of

the configuration file is JSON. See http://www.json.org/ for the format definition. Also refer to the example configuration files in the installation data directory.

2.1. Configuration Locations and Files

Starting with RCE 6.0.0, all user data is strictly separated from the RCE installation itself. Each set of user data is contained in a so-called "profile". Each profile defines what is called an RCE "instance". Each profile (and therefore, each instance) belongs to exactly one user, and each user can have multiple profiles. The default profile is located within the user's "home" directory ("/home/<user id>/" on Linux), in the ".rce/default" sub-folder.

Note

Note that ".rce" is a hidden directory; you may need to set operating-specific options to see hidden files and directories.

All manual configuration takes place in the profile's central configuration file, "configuration.json". As of RCE 6.0.0, most configuration settings only take effect on startup, so you need to restart RCE after editing it. (This will be changed in an future release.) This applies to all types of installations.

2.2. Configuration Parameters

Configuration parameters are grouped within the configuration file. Below are lists of the configuration parameters. There is one list per group.

Table 2.2. general

Configuration key	***Comment***	***Default value***
instanceName	The name of the instance that will be shown to all users in the RCE network.	"<unnamed instance>"
isWorkflowHost	If set to *true*, the local instance can be used as a *workflow host* by other RCE instances. I.e., the workflow controller can be set to this instance and the workflow data is stored there as well.	false
isRelay	If set to *true*, the local node will merge all connected nodes into a single network, and forward messages between them. This behaviour is transitive; if a relay node connects to another relay node, both networks will effectively merge into one. If set to *false* (the default value), the local node can connect to multiple networks at once without causing them to merge.	false
tempDirectory	Can be used to override the default path where RCE stores temporary files. Useful if there is little space in the default temp file location. Must be an absolute path to an existing directory, and the path must not contain spaces (to prevent problems with tools accessing such directories).	An "rce-temp" subdirectory within the user or system temp directory.

Note

IMPORTANT: When setting up a network of RCE instances, keep in mind that the RCE network traffic is currently *not encrypted*. This means that it is *not secure* to expose RCE server ports to untrusted networks like the internet. When

setting up RCE connections between different locations, make sure that they either connect across a secure network (e.g. your institution's internal network), or that the connection is secured by other means, like SSH tunnelling or a VPN.

Table 2.3. network

Configuration key	*Comment*	*Default value*
requestTimeoutMsec	The timeout (in milliseconds) for network requests that are made by the local node. If this time expires before a response is received, the request fails.	40000
forwardingTimeoutMsec	The timeout (in milliseconds) for network requests that are forwarded by the local node on behalf of another node. If this time expires before a response is received, an error reponse is sent back to the node that made the request.	35000
connections	A map of all connections that the local instance tries to establish on startup. This allows the local instance to act as a client. For each connection a unique identifier (id) must be given.	{} (an empty map in JSON format)
connections/[id]/host	IP address of the host to connect to. Host names and IPv4 adresses are permitted.	-
connections/[id]/port	Port number of the remote RCE instance.	-
connections/[id]/connectOnStartup	If set to *true*, the connection is immediately established on startup.	true
connections/[id]/autoRetryInitialDelay	The initial delay, in seconds, to wait after a failed or broken connection before a reconnect attempt is made. This configuration must be present to enable the auto-reconnect feature.	-
connections/[id]/autoRetryDelayMultiplier	A decimal-point value >= 1 that the delay time is multiplied with after each consecutive connection failure. This provides an "exponential backoff" feature that reduces the frequency of connection attempts over time. This configuration must be present to enable the auto-reconnect feature.	-
connections/[id]/autoRetryMaximumDelay	Defines an upper limit for the delay time, even when applying the multiplier would create a higher value. This can be used to maintain a minimum frequency for retrying the connection. This configuration must be present to enable the auto-reconnect feature.	-
serverPorts	A map of all server ports that the local instance registers for other instances to connect to. This allows the local instance to act as a server. For each server port a unique identifier (id) must be given.	{} (an empty map in JSON format)
serverPorts/[id]/ip	IP address to which the local instance should be bound.	-
serverPorts/[id]/port	Port number to which other instances connect to.	-
ipFilter	Allows to limit the incoming connections to a set of IP addresses.	-
ipFilter/enabled	If set to *true*, the ip filter active.	false

Configuration key	*Comment*	*Default value*
ipFilter/allowedIPs	List of IP addresses, which are allowed to connect to the instance.	[] (an empty list in JSON format)

Table 2.4. publishing

Configuration key	*Comment*	*Default value*
components	List of workflow components, which are made available to other instances within the RCE network. Workflow components are represented by their identifiers. Note: That doesn't affect integrated tools.	[] (an empty list in JSON format)

Table 2.5. componentSettings

Configuration key	*Comment*	*Default value*
de.rcenvironment.cluster	Configuration of the cluster workflow component.	-
de.rcenvironment.cluster/maxChannels	Maximum number of channels, which are allowed to be opened in parallel to the cluster server.	8
de.rcenvironment.sql	Configuration of the SQLworkflow components.	-
de.rcenvironment.sql/[id]/profiles	List of profiles which will describe the jdbc connection to a database. All configuration keys following are part of one entry of the collection	{} (an empty list in JSON format)
de.rcenvironment.sql/[id]/label	A name for a profile. This label will be shown in the configuration view of the SQL component	-
de.rcenvironment.sql/[id]/host	A url to the host computer of the database	-
de.rcenvironment.sql/[id]/database	The name of the database	-
de.rcenvironment.sql/[id]/user	The username for database access	-
de.rcenvironment.sql/[id]/password	The password of the user for database access	-
de.rcenvironment.sql/[id]/jdbc	List of jdbc configuration parameters. All configuration keys following are part of this list	{} (an empty list in JSON format)
de.rcenvironment.sql/[id]/url	Connection string of jdbc driver. Per default, it will be assembled with configuration keys above.	-
de.rcenvironment.sql/[id]/file	Absolute or relative (beginning at root-folder of RCE) path to jdbc driver file	-
de.rcenvironment.sql/[id]/driver	SQL driver classname	

Table 2.6. sshServer

Configuration key	*Comment*	*Default value*
enabled	If set to *true* the local instance acts as an SSH server.	false
ip *(deprecated alias: "host")*	The host's ip address to bind to. If you want to make the SSH server accessible from remote, you should set this to the IP of the machine's external	127.0.0.1

Configuration key	***Comment***	***Default value***
	network interface. Alternatively, you can set this to "0.0.0.0" to listen on all available IPv4 adresses, if this is appropriate in your network setup.	
port	The port number to which SSH clients can connect to.	-
accounts	A map of accounts. For each account a unique identifier (account name) must be given.	{} (an empty map in JSON format)
[account name]/ password	The password for the account. IMPORTANT: As of RCE 6.x, SSH passwords are always configured as plain text. To prevent misuse of the configured login data, any configuration file with SSH accounts *must* be secured against unauthorized reading (e.g. by setting restrictive filesystem permissions). RCE 7.x will add support for configuring passwords in a more secure manner.	-
[account name]/role	The role of the account. See 'roles' below.	-
[account name]/enabled	If set to *true*, the account is enabled.	true
roles	A map of roles. For each role a unique identifier (role name) must be given.	{} (an empty map in JSON format)
[role name]/ allowedCommandPatterns	List of commands the role is allowed to execute. Regular expressions are supported.	[] (an empty list in JSON format)

Chapter 3. Usage

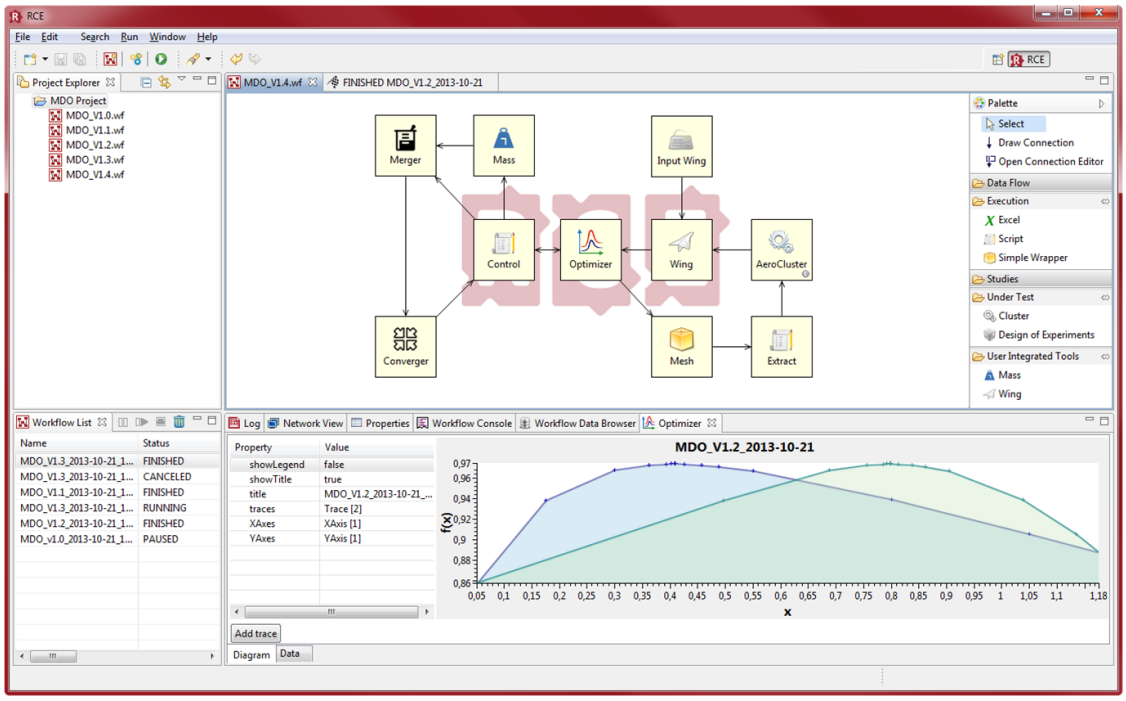
This chapter describes the main usage concepts.

1. Graphical User Interface

This section introduces the Graphical User Interface (GUI).

The GUI of RCE is composed of different views and editors (besides standard GUI elements such as the menu bar, status bar, etc.). Views can be (re-)arranged by the user. They can even be closed and opened again. Some views are closed by default, but can be opened as desired. To open a view go to: Window->Show view.

Figure 3.1. Workbench with different views and the workflow editor opened



Left hand-side:

- *Project Explorer:* View to manage projects. All relevant data including workflow files needs to be organized in projects.

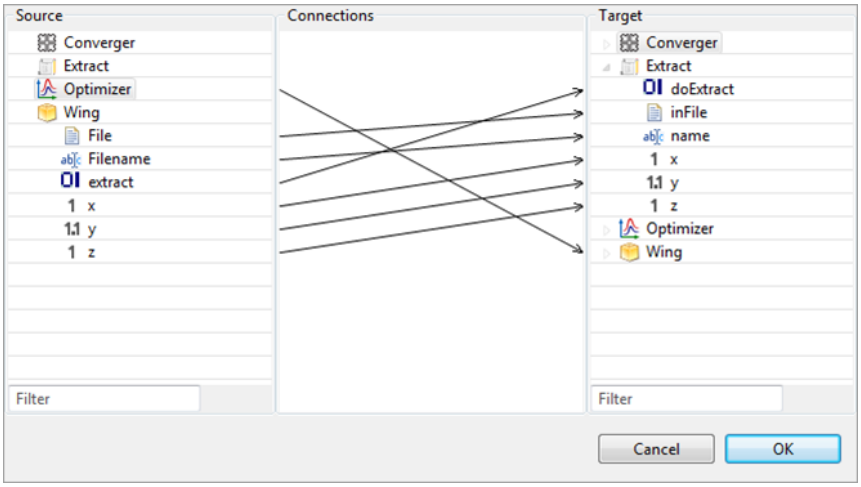
- *Workflow List:* Lists all active workflows and allows to manage them (stop, pause, resume, dispose).

Right hand side and center:

- *Workflow Editor:* Core view of RCE used to build and configure workflows.

- *Palette:* Lists all available workflow components. If RCE runs in a distributed environment this includes local as well as remote workflow components. At the top, it also provides actions for connecting workflow components. The appearing connection editor dialog will look like this. Additionally, connections of the workflow are shown in the *Properties* view at the bottom, if the background of the workflow editor is selected.

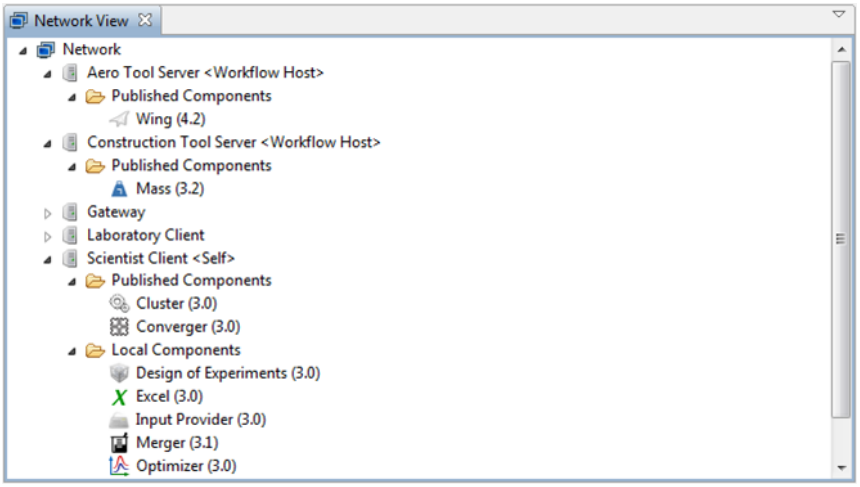
Figure 3.2. Connection Editor



Bottom:

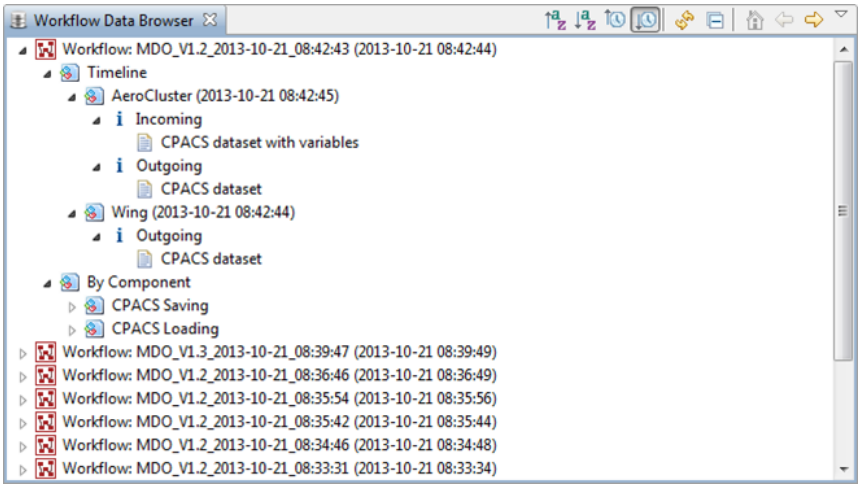
- *Log:* Shows all log output of RCE, e.g. error messages during workflow execution.
- *Network View:* Shows all RCE nodes of the distributed RCE network and their published workflow components. It also shows the outgoing connections of the own RCE instance and allows to manage them (start, stop, etc.).

Figure 3.3. Network View



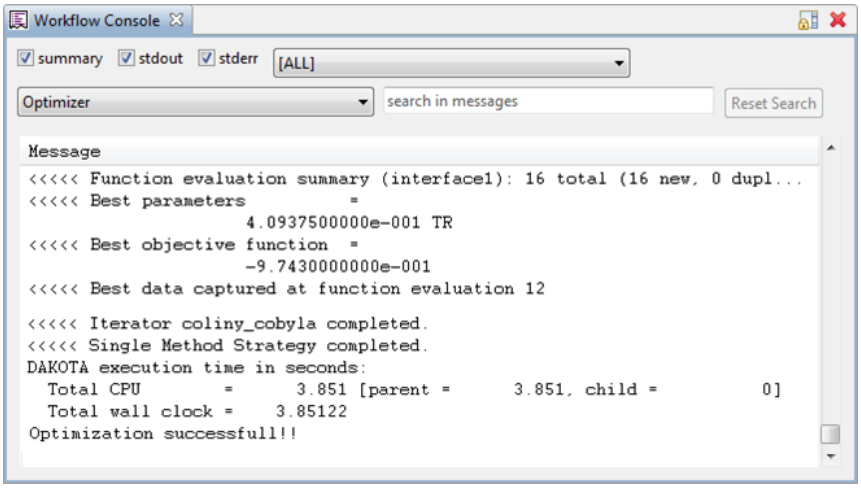
- *Workflow Data Browser:* Shows workflow related result data.

Figure 3.4. Workflow Data Browser



- *Properties:* Allows configuration of workflow components (e.g. Inputs/Outputs) if they are selected in the workflow editor. View adapts to selected workflow component.
- *Workflow Console:* Shows all native console line output of integrated tools during workflow execution. Provides full text search.

Figure 3.5. Workflow Console



2. Running and Creating Workflows

RCE is designed to execute automated workflows. The following steps will help you to execute provided example workflows and to create and execute your own workflow with RCE:

Execute provided example workflows:

- Start RCE.
- Create a project with example workflows shipped with RCE: File->New->Other... Expand item *RCE*, select *Workflow Examples Project* and click *Next*. Give the project a name of your choice (you can leave the provided default as well) and click *Finish*. In the Project Explorer on the left-hand side your project appears.

- Execute one of the example workflows. Double-click on a workflow of your choice (workflow files ends with .wf). The workflow is opened in the workflow editor. Right-click in workflow editor and click *Execute workflow* on the bottom of the menu . Confirm run configuration in the dialog by clicking *Ok* : The graphical representation of the executing workflow will be opened. Little icons indicate different states of single executing components.

- See the Workflow List for the current state of the workflow.

- See the Workflow Console for native console line output if there is some generated by a workflow.

- See the Log for log messages, e.g. info, erros, etc.

Create your own workflow:

- Open *File->New Workflow...* and follow the wizard dialog. In the end you will have a new workflow file, which is immediately opened in the workflow editor.

- Drag *Parametric Study* and *Script* from the workflow component palette on the right-hand side into the editor area.

- Configure Script component: Double-click on Script component in the editor: Properties view is opened in the bottom.

- Create a floating point (Data type: Float) input and output: Switch to *In/Outputs* tab. Add an input with name "incoming" of type Float. Add an output with name "outgoing" of type Float.

- Write the underlying Python script: Switch to *Script* tab. Clear the text area. Write

```
value = RCE.read_input('incoming')

print value

RCE.write_output('outgoing', 2*value)
```

- Configure Parametric Study: Double-click on Parametric Study component in the editor: Properties view is opened in the bottom.

- Switch to *Study Range* tab. Double-click on the entry in the table and fill in the field to perform a parametric study from 1 to 10 in 0.5 steps.

- Switch to *Parameters* tab. Add a parameter with name "param" of type Float.

- Connect the components: Select in the palette on the right-hand side: *Draw Connection*. The appearance of the cursor should change. Pull a connection from Parametric Study to Script by first clicking on Parametric Study and then on Script in the graphical editor. The connection editor opens. If the choice is unique the connection is shown automatically. Otherwise: In the connection editor add a connection from *Design Variable* of Parametric Study to *incoming* of the Script. In the connection editor drag a connection from *outgoing* of Script onto *param* of the Parametric Study.

- Well done: You just created a workflow with a parametric study from 1 to 10 in 0.5 steps. Thereby, the generated values will be doubled by the Script component and returned to the Parametric Study component as parameters to track.

- Exceute the workflow as the ones above and see Workflow Console, Workflow List, etc. for information. Double-click on Parametric Study component in the runtime workflow view and you can monitor the *param* parameter.

3. Workflow Components

Documentation regarding workflow components is integrated in RCE (as so called dynamic help). It can be found by pressing the F1 key, which opens the dynamic help view. After that, select the workflow component in the workflow editor you would like to get help for. The view will display the help content of that particalur workflow component. You can navigate through the contents by clicking the links and by using backwards and forwards.

3.1. Execution Scheduling of Workflow Components

The execution of workflows is data-driven. As soon as all of the desired input data is available, a component will be executed. Which input data is desired, is defined by the component developer (in case of native RCE workflow components), the tool integrator, and/or the workflow creator. The component developer and tool integrator decide which options are allowed for a particular component. The workflow creator can choose between these options at workflow design time. The following options exist:

Input handling

- *Constant:* The value won't be consumed during execution and will be reused in the next iteration (if there is any loop in the workflow). The workflow will fail if there is more than one value received, except for nested loops: All inputs of type *Constant* are resetted within nested loops, after the nested loop has been terminated.
- *Single (Consumed):* The input value will be consumed during execution and won't be reused in the next iteration (if there is any loop in the workflow). Queuing of input values is not allowed. If another value is received before the current one was consumed, the workflow will fail. This can guard against workflow design errors. E.g., an optimizer must not receive more than one value at one single input within one iteration.
- *Queue (Consumed):* The input value will be consumed during execution and won't be reused in the next iteration (if there is any loop in the workflow). Queuing of input values is allowed.

Execution constraint

- *Required:* The input value is required for execution. Thus, the input must be connected to an output.
- *Required if connected:* The input value is not required for execution (e.g., if a default value will be used as fall back within the component). Thus, the input doesn't need to be connected to an output. But if it is connected to an output, it will be handled as an input of type *Required*.
- *Not required:* The input value is not required for execution. Thus, the input doesn't need to be connected to an output. If it is connected to an output, the input value will be passed to the component if there is a value available at the time of execution. Values at inputs of type *Not required* cannot trigger component execution except if it is the only input defined for a component. Note: With this option, non-deterministic workflows can be easily created. Use this option carefully. If in doubt, leave it out.

Note: With RCE 6.0.0 the scheduling options changed. Below is the migration path:

- *Initial* was migrated to *Constant* and *Required*.
- *Required* was migrated to *Single (Consumed)* and *Required*.
- *Optional* was migrated to *Single (Consumed)* and *Required if connected*.

If you encounter any problems with workflows created before RCE 6.0.0, it is likely, that it affects the migration to *Single (Consumed)* instead of to *Queue (Consumed)*. We decided to migrate

conservatively to not hide any existing workflow design errors. So, if queuing of input values is allowed for an input, just change the input handling option to *Queue (Consumed)* after the workflow was updated. Another issue can affect the migration of *Optional*. If it affects an input of the script component, check the option, which let the script component execute on each new value at any of its inputs. Also check *Not required* as an alternative execution constraint option.

4. Commands

This section introduces the list of commands available for the command line and the interactive shell.

4.1. Command Line Parameters

General syntax

```
>  rce --[RCE arguments] -[RCP arguments] -[VM arguments]
```

Table 3.1. Command line arguments for RCE

Argument	Type	Description
profile "<profile id or path>"	RCE	Sets a custom profile folder to use. If only an id (any valid directory name) is given, the profile directory "<user home>/.rce/*id*" is used. Alternatively, a full filesystem path can be specified.
batch "<command string>"	RCE	Behaves like the "exec" command, but also implies the "--headless" option and always shuts down RCE after execution.
headless	RCE	Starts RCE in a headless modus without GUI. It will remain in the OSGi console and waits for user input.
exec "<command string>"	RCE	Executes one or more shell commands defined by <command string>. For the list of available commands, refer to the command shell documentation. This argument is usually used together with --headless to run RCE in batch mode. Multiple commands can be chained within <command string> by separating them with " ; " (note the spaces); each command is completed before the next is started. You can use the "`stop`" command at the end of the command sequence to shut down RCE after the other commands have been executed. However, any error during execution of these commands will cancel the sequence, and prevent the "`stop`" command from being executed. To ensure shut down at the end of the command sequence, use the `--batch` option instead of "`--exec`". As an example, `rce --headless --exec "wf run example.wf ; stop"` will execute the "example.wf" workflow in headless mode and then shut down RCE. However, if the workflow fails to start, RCE will keep running, as the "`stop`" command is never executed. To attempt execution of the workflow file, but then always shut down regardless

Argument	Type	Description
		of the outcome, use `rce --batch "wf run example.wf"` instead.
data @noDefault	RCP	Set the default workspace location to empty
consoleLog	RCP	Logs everything for log files on the console as well.
console	RCP	Runs RCE with an additional OSGi console window, which allows you to execute RCE shell commands. See the Command Shell documentation for more information.
Deprecated: console <port>	RCP	Specify the port that will be used to listen for telnet connections. (*NOTE:* this access is insecure; configure SSH access instead)
clean	RCP	Cleans before startup
vmargs	VM	Standard JVM arguments
Dde.rcenvironment.rce.configuration.dir=<insert-config-path>	VM	Sets the configuration directory
Dcommunication.overrideNodeId =<some-id>	VM	Sets the local node id, overriding any stored value. This is mostly used for automated testing. Example: "-Dcommunication.overrideNodeId=a96db8fa762d59f2d2782f3e5e9662d4" *Note*: This setting's format will most likely change in 5.0.0.
Dcommunication.uploadBlockSize=<block size in bytes>	VM	Sets the block size to use when uploading data to a remote node. This is useful for very slow connections (less than about 10 kb/s) to avoid timeouts. The default value is 262144 (256 kb). Example: "-Dcommunication.uploadBlockSize=131072" - sets the upload block size to 128kb (half the normal size) *Note*: This setting's format will most likely change in 5.0.0.

4.2. Command Shell

RCE provides an integrated shell (sometimes referred to as "console") for executing commands. It can be accessed in three different ways:

- Start RCE with the "-console" command-line option, or add "-console" to the rce.ini file before starting; this will open an OSGi console window. Due to the nature of an OSGi console, all RCE commands must be prefixed with "rce". For example, type "rce help" to show the available commands.

- *Deprecated:* Start RCE with the "-console <port>" command-line option; this will accept telnet OSGi console sessions on that port. As with the "-console" option, RCE commands must be prefixed with "rce" (for example, type "rce help").

  Note that this option is *insecure*, as there is no authentication nor encryption, so it should only be used in fully trusted networks. Whenever possible, use the SSH console (see below) instead .

- Configure SSH access. To do so, refer to Section Configuration Parameters. After RCE has started, you can access the shell on the configured port with a standard SSH client. On Windows systems, the "putty" software works well as a client.

  As this option creates a pure RCE shell (as opposed to the OSGi consoles created above), you can enter RCE commands without a prefix - for example, just type "help" to list the available commands. Note that to avoid confusion, adding a "rce" prefix still works, but it is not necessary.

The following table lists some shell commands; more documentation coming soon.

Table 3.2. Shell Commands

Argument	Description
help	Lists all available commands.
components	Short form of "components list".
components list	Lists components published by reachable RCE nodes.
cn	Short form of "cn list".
cn add <target> ["<description>"]	Adds a new network connection. (Example: cn add activemq-tcp:rceserver.example.com:20001 "Our RCE Server")
cn list	Lists all network connections, including ids and connection states.
cn start <id>	Starts/Connects a READY or DISCONNECTED connection (use "cn list" to get the id).
cn stop <id>	Stops/Disconnects an ESTABLISHED connection (use "cn list" to get the id).
net	Short form of "net info".
net filter	Shows the status of the IP whitelist filter.
net filter reload	Reloads the IP whitelist configuration.
net info	Lists all reachable RCE nodes.
restart	Restarts RCE.
shutdown	Shuts down the local RCE instance.
stop	Shuts down the local RCE instance (alias of "shutdown").
version	Shows version information.
wf	Short form of "wf list"
wf list	Lists all current workflows, their states and execution ids.
wf details <workflow execution id>	Shows details about one workflow.
wf cancel <workflow execution id>	Cancels a running or paused workflow.
wf dispose <workflow execution id>	Disposes a finished, cancelled or failed workflow.
wf pause <workflow execution id>	Pause a running workflow.
wf resume <workflow execution id>	Resume a paused workflow.
wf run [--dispose <onfinished|always|never>] [--compact-output] [-p	Executes the given workflow file and waits until it has completed. Workflow file paths containing spaces must be enclosed in double quotes ("..."). The "--dispose" option defines the disposal behavior after workflow completion. Disposing a workflow releases certain resources that may or may not be used after it has finished, for example data to be visualized in

Argument	Description
<placeholder value file>] <workflow file>	components' runtime views. Data that is stored in the data management is not affected by disposal. The default of this setting is "onfinished": The workflow is disposed if it terminates in state "Finished" (which means normal completion without errors), otherwise it is left unchanged for inspection. The "--compact-output" option reduces this command's output as much as possible, which is intended to simplify scripted calls of this command. The first line printed will either be the workflow's assigned id if the start was successful, or a text starting with `"Error "` if the workflow could not be started. If (and only if) the start was successful, a second like will be printed once the workflow has terminated. The pattern of this second line is "`<workflow id>: <final state>`". The "-p" option can be used to define a placeholder value file (see below).
wf verify [--dispose <onfinished|always|never>] [--pr <parallel runs>] [--sr <serial runs>] [-p <placeholder value file>] --basedir <directory> <workflow file> [<workflow file> ...]	Runs several workflows and creates a summary of which ones failed and succeeded. The "--pr" option defines how often the workflow is started in parallel. The "--sr" options defines how often the workflow is started in serial. E.g. "--pr 5 --sr 3" starts the workflow three times with five in parallel. If "*" is used with the "--basedir" option or multiple workflow filenames are passed, "--pr" and "--sr" are applied for each of the workflows. For the "--dispose" and "-p" options refer to "wf run" above. The "--basedir <directory>" parameter specifies the directory containing the workflow files. File paths containing spaces must be enclosed in double quotes ("..."). The second parameter defines the workflow's filenames. Using "*" as workflow file runs all non-backup workflows in the basedir. Workflow file paths containing spaces must be enclosed in double quotes ("...").

4.2.1. Configuration Placeholder Value Files

Some workflow components use placeholders for configuration values. The values for the placeholders are defined at workflow start. When executing workflows from the command line (e.g. in headless or batch mode), the placeholder's values must be defined in a file, which will be passed to the command with the -p option. Placeholder value files have following format:

```
{

<component id>/<component version> : {

<configuration placeholder id> : <configuration value>

}

<component id>/<component version>/<component instance name> : {

<configuration placeholder id> : <configuration value>

}

}
```

Note

Every id and every value must be in enclosed in double quotes ("...").

The `component id` is the id string of a component (e.g. de.rcenvironment.script), the `component version` is the version of the component that is used in the workflow (e.g. 3.4).

There are two ways of defining values for configuration placeholders: per component type and per component instance. When defined per component type, the id and version must be specified (e.g. "de.rcenvironment.script/3.4"). When defined per component instance the component id, component version, and the name of the component in the workflow must be specified (e.g. "de.rcenvironment.script/3.4/MyScript"). In both cases, the `configuration placeholder id`, which is the name of the configuration placeholder, and the actual `configuration value` must be specified.

Component instance values override component type values.

Note

It is possible to mix component type and component instance values.

Below is an example placeholder value file, which defines one placeholder value (component type) for the script component, one placeholer value (component type) for the input provider component and a placeholder value (component instance) for a specified input provider component of the workflow:

```
{

"de.rcenvironment.script/3.4": {

"pythonExecutionPath": "C:/Python/python.exe"

},

"de.rcenvironment.inputprovider/3.2": {

"inputFile": "C:/input/globalInputFile.txt"

},

"de.rcenvironment.inputprovider/3.2/Provider 1" : {

"inputFile": "C:/input/Provider1.txt"

}

}
```

The following table lists components and their configuratuon placeholders.

Table 3.3. Components and their configuration placeholders

Component	Component id and version	Configuration placeholders
Cluster	de.rcenvironment.cluster/3.2	authuser - user name authphrase - password (base64 encoded)
Input Provider	de.rcenvironment.inputprovider/3.2	<output name> - value of output

Component	Component id and version	Configuration placeholders
Output Writer	de.rcenvironment.outputwriter/1.1	targetRootFolder - path to target root folder
Script	de.rcenvironment.script/3.4	pythonExecutionPath - path to the Python executable (only required if Python is set as script language)

5. Integration of External Tools

5.1. Basic Concepts

The Tool Integration concept of RCE is used to integrate external tools for calculation or other things into RCE and use them in a workflow as a component. The tools must fulfill some requirements:

- The tool must be callable via command line
- It must have a non-interactive mode which is called via command line
- Input for the tool must be provided trough command line arguments or files

If these requirements are fulfilled, a configuration file that is used for the integration can be created.

This can be done with the help of an integration wizard if you use RCE with a graphical user interface. This wizard can be found in the menu *File -> Integrate a Tool*. Everything that is needed for an integration will show up in this wizard. Required fields are marked with an asterisk (*). When the wizard is finished and everything is correct, the integrated tool will automatically show up in the workflow editor palette.

Note

The wizard has a dynamic help, which is shown by clicking on the question mark on the bottom left or by pressing F1. It will guide you through every page of the wizard.

5.2. Copying of Integrated Tools

When a component is created in the integration wizard, a configuration file is created.

All configuration files from the tool integration are stored in the directory

```
<profile folder>/integration/tools/
```

In this directory, there is a separation between different kinds of integration realized through one subdirectory for each. The `common` folder is always existent.

In these subdirectory, the integrated tools are stored, again separated through a subdirectory for each. The name of the directory equals the name of integration of the tool.

If a tool integration should be copied to another RCE instance or another machine, the directory of the tool must be copied, containing a `configuration.json` and some optional files. It must be put in the equivalent integration type directory of the target RCE instance. After that, RCE automatically reads the new folder and if everything is valid, the tool will be integrated right away.

5.2.1. Publishing a Component on a Server Instance in Headless Mode

An integrated tool on a server is not neccessarily visible for other user in the same network. Publishing it means to make it available for all others, but they are not allowed to change the tool's integration. If a component should be published, the `published.conf` file in the integration kind subdirectory must be edited. In this file, the tool's directory name (or absolute path) must be written in a single line. After saving the file, RCE again reads the file automatically and the component will be published in the network.

Note

Using a graphical editor to edit the `published.conf` or a tool configuration file **might** not work as expected. It it possible, that the tool or the publishing status is not updated when saving the tool. This problem did not occur with headless editors.

For example the emacs graphical editor does not work for editing the files, but vi does.

5.3. Integration of CPACS Tools

5.3.1. Additional concepts of CPACS Tool Integration

Extending the common Tool Integration concept, the CPACS Tool Integration has some additional features.

- Parameter Input Mapping (optional): Substitutes single values in the incoming CPACS content, based on an XPath configured at workflow design time as a dynamic input of the component
- Input Mapping: Generates the tool input XML file as a subset of the incoming CPACS file XML structure, specified by a mapping file
- Tool Specific Input Mapping (optional): Adds tool specific data to the tool input file, based on a mapping file and a data XML file
- Output Mapping: Merges the content of the tool output XML file into the origin incoming CPACS file, based on a mapping file
- Parameter Output Mapping (optional): Generates output values as single values of the CPACS result file, based on an XPath configured at workflow design time as a dynamic output of the component
- Execution option to only run on changed input: If enabled, the integrated tool will only run on changed input. Therefore the content of the generated tool input file is compared to the last runs content. Additionally the data of the static input channels are compared to the previous ones.

All the features listed above can be configured in the tool integration wizard on the dedicated *CPACS Tool Properties* page.

The mappings can be specified by XML or XSLT as shown in the following examples. RCE differentiates between these methods in accordance to the corresponding file extension (.xml or .xsl).

Example for an input or toolspecific XML mapping :

```
<?xml version="1.0" encoding="UTF-8"?>
<map:mappings xmlns:map="http://www.rcenvironment.de/2015/mapping"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <map:mapping>
```

```
        <map:source>/cpacs/vehicles/aircraft/model/reference/area</map:source>
        <map:target>/toolInput/data/var1</map:target>
    </map:mapping>

</map:mappings>
```

And input or toolspecific XSLT mapping:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="cpacs_schema.xsd">
 <xsl:output method="xml" media-type="text/xml" />
 <xsl:template match="/">
  <toolInput>
   <data>
    <var1>
     <xsl:value-of select="/cpacs/vehicles/aircraft/model/reference/area" />
    </var1>
   </data>
  </toolInput>
 </xsl:template>
</xsl:stylesheet>
```

Example of an output XML mapping:

```
<?xml version="1.0" encoding="UTF-8"?>
<map:mappings xmlns:map="http://www.rcenvironment.de/2015/mapping"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <map:mapping mode="delete">
        <map:source>/toolOutput/data/result1</map:source>
        <map:target>/cpacs/vehicles/aircraft/model/reference/area</map:target>
    </map:mapping>

</map:mappings>
```

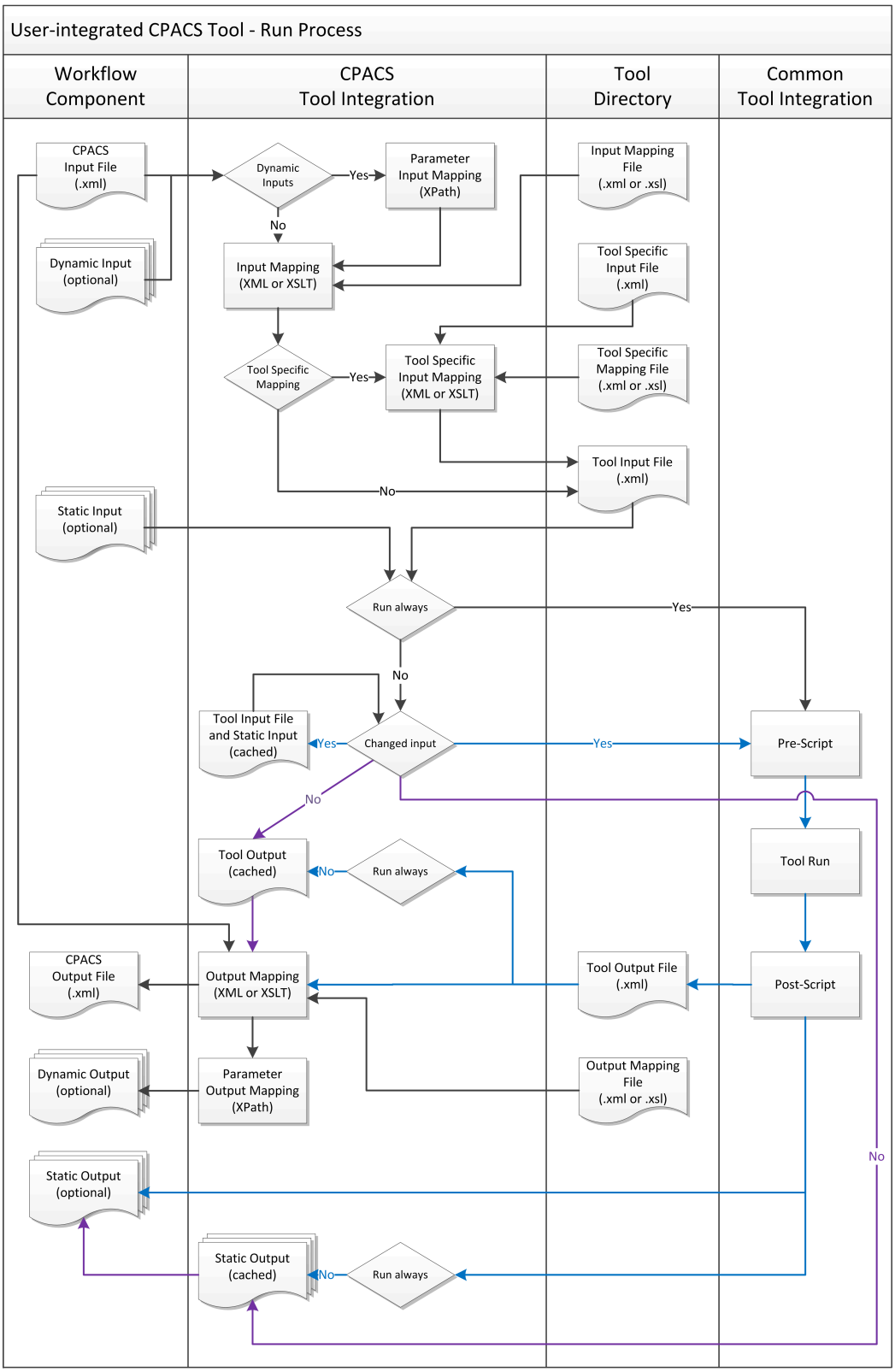
And output XSLT mapping:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" exclude-result-
prefixes="xsi">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <!--Define Variable for toolOutput.xml-->
  <xsl:variable name="toolOutputFile" select="'./ToolOutput/toolOutput.xml'"/>
  <!--Copy complete Source (e.g. CPACSInitial.xml) to Result (e.g.
 CPACSResult.xml)-->
  <xsl:template match="@* | node()">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
  <!--Modify a value of an existing node-->
    <xsl:template match="/cpacs/vehicles/aircraft/model/reference/area">
    <area>
    <xsl:value-of select="document($toolOutputFile)/toolOutput/data/result1"/>
    </area>
    </xsl:template>
</xsl:stylesheet>
```

Please ensure to use the proper namespace for map (xmlns:map="http://www.rcenvironment.de/2015/mapping") in XML mapping files.

The figure below illustrates how the additional features are used in the run process of an user-integrated CPACS tool.

Figure 3.6. Run process of an user-integrated CPACS Tool



5.3.2. Integrate a CPACS Tool into a Client Instance

1. Start RCE as Client
2. Open the *Tool Integration Wizard* by clicking the *Integrate Tool* in the *File* menu.

Note

You will always find further help by clicking the *?* on the bottom left corner on each page of the wizard or by pressing *F1*.

3. Choose the option *Create a new tool configuration from a template*.

Note

The CPACS templates delivered with RCE are designed to match the conventions of the old CPACS tool wrapper (respectively ModelCenter tool wrapper). Most of the properties are preconfigured and do not need to be changed.

4. Select one of the *CPACS* templates.
   Click *Next*.

5. Fill in the *Tool Description* page.
   Click *Next*.

6. On the *Inputs and Outputs* page you will find preconfigured static in- and outputs, that will match the old tool wrapper conventions. If your tool needs additional in- or outputs, feel free to configure.
   Click *Next*.

7. Skip the page *Tool Properties* by clicking *Next* since it is not relevant for tools that match the conventions of the old CPACS tool wrapper.

8. Add a launch setting for the tool by clicking the *Add* button on the *Launch Settings* page. Configure the path of the CPACS tool and fill in a version, click *OK*. If you would like to allow users of your tool to choose that the temp directory won't be deleted at all after workflow execution, check the property *Never delete working directory(ies)*. Not to delete the working directory can be very useful for users for debugging purposes, at least if they have access to the server's file system. But this option can result in disc space issues as the amount required grows continuously with each workflow execution. It is recommended to check that option during integrating the tool and uncheck it before publishing the tool. If you like to publish your tool to the entire network right after the integration, check *Publish component after integration*. Note that you can publish your tool at any later point in time.
   Click *Next*.

9. The *CPACS Tool Properties* are preconfigured to match the folder structure defined for the old CPACS tool wrapper. In most cases you do not have to change this configuration. If you are using XSLT mapping, please select the corresponding mapping files. If your tool does not work with static tool specific input, please deselect this property.
   Click *Next*.

10. In the *Execution command(s)* tab on the *Execution* page, you need to define your execution command itself as well as optional pre and post commands. Commands will be processed sequentially line by line. An example for a typical Windows command including pre and post commands will look like the following:

```
rem pre-command
pre.bat

rem tool-execution
YourTool.exe ToolInput/toolInput.xml ToolOutput/toolOutput.xml

rem post-command
post.bat
```

11. Click *Save and activate* and your tool will appear immediately in the palette and is be ready to use.

12. If not already done, do not forget to publish your tool after testing it locally. Therefore, edit your tool configuration by clicking *Edit Tool* in the *File* menu, navigate to the page *Launch Configuration* and check the property *Publish component after integration*. Click *Save and activate*. To check if

your tool is successfully published to the RCE network open the tab *Network View* at the bottom and checkout *Published Components* after expanding the entry of your RCE instance.

5.3.3. Integrate a CPACS Tool into a Server Instance in Headless Mode

The way to integrate a CPACS tool on a server running RCE in headless mode is as follows: Perform the steps to integrate a CPACS tool on a client instance and make sure that the path of the CPACS tool configured on the *Launch Settings* page (step 8) matches the absolute tool path on your server system. Afterwards, you will find the configuration files inside your rce profile folder at the following location:

```
/integration/tools/cpacs/[YourToolName]
```

Copy the folder `[YourToolName]` to the same location inside the profile folder running with your headless server instance. Edit the file `published.conf` in the cpacs directory by adding your tool's directory name as a single line. If the server instance is already running, your tool will be available immediately after saving the file (cf. section "Publishing a component in headless mode").

6. Remote Tool and Workflow Access

RCE provides an interface that allows external applications to access and run single tools or complete workflows within RCE instances. This allows existing applications to make use of RCE's features (like network distribution, data management, logging, ...) when integrating the application itself is not possible or desirable. For example, applications that are based on frequent user interaction are not a good fit for being run as part of an automated workflow, but may still want to use some of RCE's features.

This section describes the Remote Access feature from an end-user's perspective. Using scripts and templates contained in the RCE distribution, this will guide you through the creation of a simple example, which you can expand to build your own solutions.

6.1. Basic Concepts

The basic idea of the Remote Access interface is that the external application opens an SSH connection to RCE, and initiates the tool or workflow execution via text commands. Input and output data, as well as log files, are transfered using SCP. To demonstrate this interface, and how to interact with it, example script files are provided for Windows and Linux. This documentation explains how to set up and use these scripts.

Note that these scripts are not the Remote Access interface itself, but only prototype examples of how to use it. Native command-line tools and system libraries (e.g. Windows .dll and Linux .so files) are under development, and will be available soon.

6.2. Prerequisites

Except for RCE (and the Java installation it needs to run), you need just one more piece of software to use the Remote Tool Access example. On Linux, the "sshpass" package is required. (This assumes that "ssh" and "scp" commands are already present on the system.)

Note

To install the "sshpass" package, check if your Linux distribution's package manager provides it. On Debian, Ubuntu or Mint systems, for example, you can install it by running the command `sudo apt-get install sshpass`

in a terminal. On other distributions, like CentOS or SUSE, you may need to compile and install it from source. To do so, download the source from `http://sourceforge.net/projects/sshpass/files/sshpass/1.05/`, unpack it, and run the standard `./configure && make && make install` sequence as root. (Please note that these steps are only required for the shell script examples to work. It is not required for the Remote Access interface itself.)

6.3. Tool vs. Workflow Execution

In RCE 5.0.0, the "remote access" feature was able to invoke a single integrated tool. Starting with RCE 5.1.0, arbitrary user-defined workflows can be executed. The following sections describe each approach separately.

6.4. Setting up the Single Tool Execution Example

These steps will guide you through the configuration of an integrated tool, and will show you how to access it using the provided example scripts. Using this example, you will be able to easily integrate your own text-based tools into RCE and invoke them using the Remote Access interface.

- Download and unpack/install an RCE distribution (version 6.0.0 or higher).
- Run RCE and select the "Help > Configuration Information" menu option. From the list of configuration examples, double-click "Remote Access Server" to open it. Mark all text (Ctrl-A) and copy it to the clipboard (Ctrl-C).
- Select the "Help > Open Configuration File" menu option, select all text (Ctrl-A) and paste the copied text from the example configuration.
- Save the file, and select "File > Restart" to apply the new configuration. The Remote Access SSH interface is now running, with a pre-configured SSH account of "ra_demo", password "ra_demo".

Note

IMPORTANT: "ra_demo" is just an example account for testing within a secure network. Create a new account with a better password before using the Remote Access interface in production. Publishing tools using this default account is a potential security risk.

- Define an example tool using the following steps.
  - Select the "File > Integrate Tool" menu option.
  - In the dialog that has opened, select "Create a new configuration from a template" and choose "Remote Tool Access" from the list. Click "Next >".
  - On the "Tool Description" page, enter a name for the example tool (e.g. "test"). Click "Next >".
  - You can leave the next two wizard pages as they are. Skip them by clicking "Next >" twice.
  - You should be on the "Launch Settings" page now. Click "Add" on the right-hand side. On the dialog that opens, you normally enter a version number describing your tool's release number and the path where it is installed. For this example, just enter an arbitrary version and select an arbitrary directory (for example, a temporary files directory on your computer). Click "Ok" and then "Next >" to proceed.

Note

After completing this Remote Access example, refer to the "Tool Integration" section on how to integrate your actual tools.

- On the "Execution" page, enable the command section that matches your operating system. This is where you define the commands that execute your tool. For this example, just leave the commands as they are. Click "Save and Activate" to complete the wizard.

- You now have an example tool that can be accessed with the "Remote Access" feature. To do so, navigate to the "examples/remote_access" sub-folder in the RCE folder. Copy the run-tool.sh file to an empty folder.

- Create a folder named "input" next to the run-tool.sh file. This folder will be uploaded and made available to the integrated tool. Copy some test files into it.

- Open the run-tool.sh file in a text editor. Enter the tool id and version you entered in the wizard in place of the "###" placeholders. Optionally, set something for the parameter placeholder as well. Save and exit the file.

- Open a terminal in the current folder.

- Make the shell script executable by running `chmod +x run.sh` and then start the demo with `./run-tool.sh` You should see some progress output now.

- When using the Remote Access feature for the first time, there may be a security warning to verify the identity of the RCE instance you are connecting to. Confirm the question by entering "y" and pressing Enter. This answer is saved, so the question should not appear again for the same RCE instance.

- After the tool run has finished, there should be two new folders next to the "input" folder, named "output" and "logs". "output" contains files generated by the integrated tool, while "logs" contains generall process information. Currently, it should contain a single file named "workflow.log" which provides detail information about the steps that were executed.

Note

Make sure to delete or rename the "output" and "logs" folders before running the Tool Access script again, as they will interfere with the next run's downloads.

- To get an impression of how this feature interacts with existing RCE features, you can examine several areas within the RCE instance.

  - Open the "Workflow List" view in RCE and watch it while the "run-tool" script is executing. After a short preparation time where the input data is uploaded, you will see the automatically generated workflow containing the tool being executed. It will disappear automatically if it finishes successfully; if it fails, it will remain in the list for review.

  - Open the "Workflow Console" view; if the tool produced any output, it should be visible there.

  - Open the "Workflow Data Browser" and refresh it; there should be a "Remote_Tool_Access-..." workflowe entry matching the remote tool run. When you expand this entry, you should see the uploaded content of the input folder, the generated output folder, any generated text output (in the "Execution Log" folder), and the exit code of the tool process (also in the "Execution Log" folder).

6.5. Setting up the Workflow Execution Example/ Template

These steps will guide you through the creation of a remote-executable workflow, and will show you how to invoke it using the provided example scripts.

- Download and unpack/install an RCE distribution (version 6.0.0 or higher).
- Run RCE and select the "Help > Open Configuration Information" menu option. From the list of configuration examples, double-click "Remote Access Server" to open it. Mark all text (Ctrl-A) and copy it to the clipboard (Ctrl-C).
- Select the "Help > Open Configuration File" menu option, select all text (Ctrl-A) and paste the copied text from the example configuration.
- Save the file, and select "File > Restart" to apply the new configuration. The Remote Access SSH interface is now running, with a pre-configured SSH account of "ra_demo", password "ra_demo".

Note

IMPORTANT: "ra_demo" is just an example account for testing within a secure network. Create a new account with a better password before using the Remote Access interface in production. Publishing tools using this default account is a potential security risk.

- As a first example we are going to execute the umodified "Remote_Workflow_Access_Template" workflow file in the Workflow Examples Project. If you haven't created this project already, right-click in the Project Explorer on the left side, and choose "New > Workflow Examples Project", and choose a name for it. The template file is contained within the project folder. To get an impression of the basic setup, open the template workflow file. You will see an Input Provider on the left side with two outputs: one Directory and one ShortText. On the right side, there is an Output Provider with a single Directory input. These are the points where the Remote Workflow Access feature sends the provided input files (as a Directory) and parameters (as a ShortText) into your workflow, and collects the final output files (as a Directory).
- As a security measure, you need to explicitly publish your workflow to allow remote access to it. This is done via a console command at this time; future RCE versions will most likely add a option to do this from the GUI. To issue this command, open the "Command Console" view (if it is not already visible) by selecting "Windows > Show View > Other" from the menu, and then double-clicking "Command Console" in the "RCE" section.
- Right-click your workflow file in the "Project Explorer" and select the "Properties" entry at the bottom of the popup menu. Mark the value of the "location" entry with the mouse, and press Ctrl +C to copy the full path to the workflow file to the clipboard.

Note

This step demonstrates how the get the path of a workflow file in the current workspace, but you can use workflow files that are located anywhere on your system.

- To make the workflow available for remote execution, enter the command `ra-admin publish-wf "<workflow file>" <id>` in the command window. Press Ctrl-V in place of *<workflow file>* to insert the path to your workflow file there. For *<id>*, choose a string (without whitespace) that callers can use to execute the workflow. Press "enter" to execute the command. The workflow file will be inspected, and you will either see a message describing what is missing, or a message that the workflow was successfully published. Fix any errors until the workflow is published.

Note

Starting with RCE 6.2.0, published workflows are persistent by default, so they will still be available after the local RCE instance is restarted. Use the `ra-admin unpublish-wf <id>` command to remove a published workflow from remote access.

To publish a workflow for the current RCE instance's life-time only, use the `-t` option: `ra-admin publish-wf -t "<workflow file>" <id>`.

- If some of the workflow's components use placeholders for configuration values, you can use the `-p` option to specify a placeholder values file. The structure of placeholder value files is explained in Section 4.2.1, “Configuration Placeholder Value Files”. Placeholder files can used with both persistent and non-persistent workflows (see above).

  Example: `ra-admin publish-wf -p myPlaceholderValues.json myWorkflowFile.wf myPublishId`

- You now have a workflow file that can be executed using the "Remote Access" feature. To do so, navigate to the "examples/remote_access" sub-folder in the RCE installation folder. Copy the run-wf.sh file to an empty folder.

Note

If you don't know where your RCE installation folder is, use "Help > Open Configuration Information".

- Create a folder named "input" next to the run-wf.sh file. This folder will be uploaded and made available to the integrated tool. Copy some test files into it.

- Open the run-wf.sh file in a text editor. Enter the workflow's `<id>` chosen above in place of the "###" placeholders. Optionally, set something for the parameter placeholder as well. Save and exit the file.

- Open a terminal in the current folder.

- Make the shell script executable by running `chmod +x run-wf.sh` and then start the demo with `./run-wf.sh` You should see some progress output now.

- When using the Remote Access feature for the first time, there may be a security warning to verify the identity of the RCE instance you are connecting to. Confirm the question by entering "y" and pressing Enter. This answer is saved, so the question should not appear again for the same RCE instance. If you receive a warning that the server's SSH key has changed (e.g. after reinstalling RCE), follow the command's instructions on how to reset the stored key.

- After the workflow run has finished, there should be two new folders next to the "input" folder, named "output" and "logs". "output" contains files generated by the integrated tool, while "logs" contains generall process information. Currently, it should contain a single file named "workflow.log" which provides detail information about the steps that were executed.

Note

Make sure to delete or rename the "output" and "logs" folders before running the Tool Access script again, as they will interfere with the next run's downloads.

- To get an impression of how this feature interacts with existing RCE features, you can examine several areas within the RCE instance.

  - Open the "Workflow List" view in RCE and watch it while the "run-wf" script is executing. After a short preparation time where the input data is uploaded, you will see the workflow being executed. It will disappear automatically if it finishes successfully; if it fails, it will remain in the list for review. You can also double-click on a running or workflow to monitor its execution.

- Open the "Workflow Console" view; if the tool produced any output, it should be visible there.
- Open the "Workflow Data Browser" and refresh it; there should be an entry for the Remote Access workflow. When you expand this entry, you should see the uploaded content of the input folder, the generated output folder, any generated text output (in the "Execution Log" folder), and the exit code of the tool process (also in the "Execution Log" folder).

6.6. Building Your Own Remote Access Workflow

After running the example/template workflow as described in the previous section, you can proceed to building your own actual workflow.

As described above, open the "Remote_Workflow_Access_Template" workflow file. You will see an Input Provider on the left side with two outputs: one Directory and one ShortText. On the right side, there is an Output Provider with a single Directory input. These are the points where the Remote Workflow Access feature sends the provided input files (as a Directory) and parameters (as a ShortText) into your workflow, and collects the final output files (as a Directory). You should leave these two components unchanged, and place your workflow logic between them. The Script component in the middle is just a placeholder - unless you need a Script component anyway, you can just delete it.

There are two basic approaches to building your workflow:

- Either you start with the template, and build your workflow between the two standard components. This is straight-forward, but means that you cannot test run the workflow from the RCE GUI (as the Input Provider will fail), but have to use the Remote Access feature to test it.
- The other approach is to build your workflow normally, where you add an Input Provider and Output Writer with the same outputs and inputs as the ones in the template (but with standard configuration values). You can then test (and if needed, modify) your workflow from the GUI until it behaves as it should. Then, mark all components *except* the Input Provider and Output writer in your workflow, and select "Copy" from the right-click menu. Switch to the template file, click an empty area, and select "Paste" from the right-click menu. Then, connect the two template components (Input Provider and Output Writer) as in your original workflow.

Note

(Advanced Usage) You can also build your workflow in the template file, add your own Input Provider and Output Writer, and use the new "Enable/Disable Component" feature to toggle between them for testing and Remote Access usage. As this requires some helper components to work, this is not recommended for your first example, but may be a useful trick to keep in mind.

After you have finished building your workflow, the process of publishing and executing it is the same as described above for the unmodified template file. Please note that publishing your workflow for remote execution automatically creates an (invisible) copy of it. Modifications you make to your workflow file are not published right away. Once you have made the changes you want to publish, run the same "ra-admin publish-wf" command again to update the published version.

Note

Tip: To repeat a previous command, presse the "up arrow" key in the Command Console window.